Crocker

Network Working Group
Request for Comments: 11

Implementation of the HOST - HOST Software Procedures in GORDO

G. Deloche, UCLA
1 August 1969

TABLE OF CONTENTS

1. INTRODUCTION

This technical note concentrates upon (1) the HOST-HOST procedures and (2) the implementation of the corresponding programs in GORDO (Operating System of the UCLA HOST).

The first section is closely related to the BBN reports No. 1822 and 1763[1] and specifies the HOST functions for exchanging messages. It mostly deals with links and connections, message structure, transactions, and control messages.

The second section is software oriented; it explains how the HOST functions are implemented and integrated into GORDO. It is involved with data structures, programs, buffers, interrupt processing, etc.

[1] Parts of this section are taken from or referred to those reports.

2. HOST-HOST PROCEDURES

2.1 Generalities

The basic idea is that several users, at a given HOST, should simultaneously be able to utilize the network by time-sharing its physical facilites.

This implies that within each HOST operating system, there must exist a special program that multiplexes outgoing messages from the users into the network and distributes incoming messages to the appropriate users. We will call this special program the Network program.

2.2 Links and Connections (See figure 1)

2.2.1 Definitions

It is convenient to consider the Network as a black box - a system whose behavior is known but whose mechanisms are not - for communicating messages between remote users rather than between pairs of HOST computers.

(a) Logical connections

We define a logical connection as being a communication path linking two users at remote $HOST_s$.

With that concept, a user (user program) in a HOST computer can (1) establish several logical connections to any remote HOST users, and (2) send or receive messages over those connections.

Connections appear to users as full duplex.

One of the purposes of the Network program is to serve the users in establishing, identifying, and maintaining these connections.

(b) Logical links

Each logical connection is made of a pair of directional links: one for transmitting, the other for receiving.

Those links, called logical links, are established by the Network programs and used by them.

Note here that users are only interested in connections and are completely unaware of links. Relationships between links and connections are carried out by the Network program.

One of the advantages to define a connection as a pair of directional links is that a HOST will have the capability to loop himself through its IMP (it opens a connection to himself). This feature can be useful for debugging purposes.

Further on through this paper we will not use any more the attribute logical when referring either to links or connections.

2.2.2 Connection types

In order to reach a high flexibility in utilizing the Network there is advantage to classify the connections.

Three types of connections are distinguished: (a) control connection, (b) primary connection, and (c) auxiliary connection.

(a) Control connection

This connection has a special status and is unique between a pair of $HOST_s$, e.g., if the Network includes x $HOST_s$, there are at most x control connections issued from one HOST.

This connection is used by remote Network programs for passing control messages back and forth. Control messages are basic to the establishment/deletion of standard connections. (See 2.4.2)

Note here that this control connection is the only connection which is ignored by the HOST users.

Let us describe now the standard connections.

(b) Primary connection

These connections connect remote users.

A primary connection:

* Is unique between a pair of users and is the first to be

established.

* Is "teletype-like", i.e.:

  - ASCII characters are transmitted;

  - Echos are generated by the remote HOST;

  - The receiving $HOST_s$ scan for break characters;

  - The transmission rate is slow (less than 20 characters/sec).

* Is mainly used for transmitting control commands, e.g., for log-in into a remote HOST operating system.

(c) Auxiliary connection

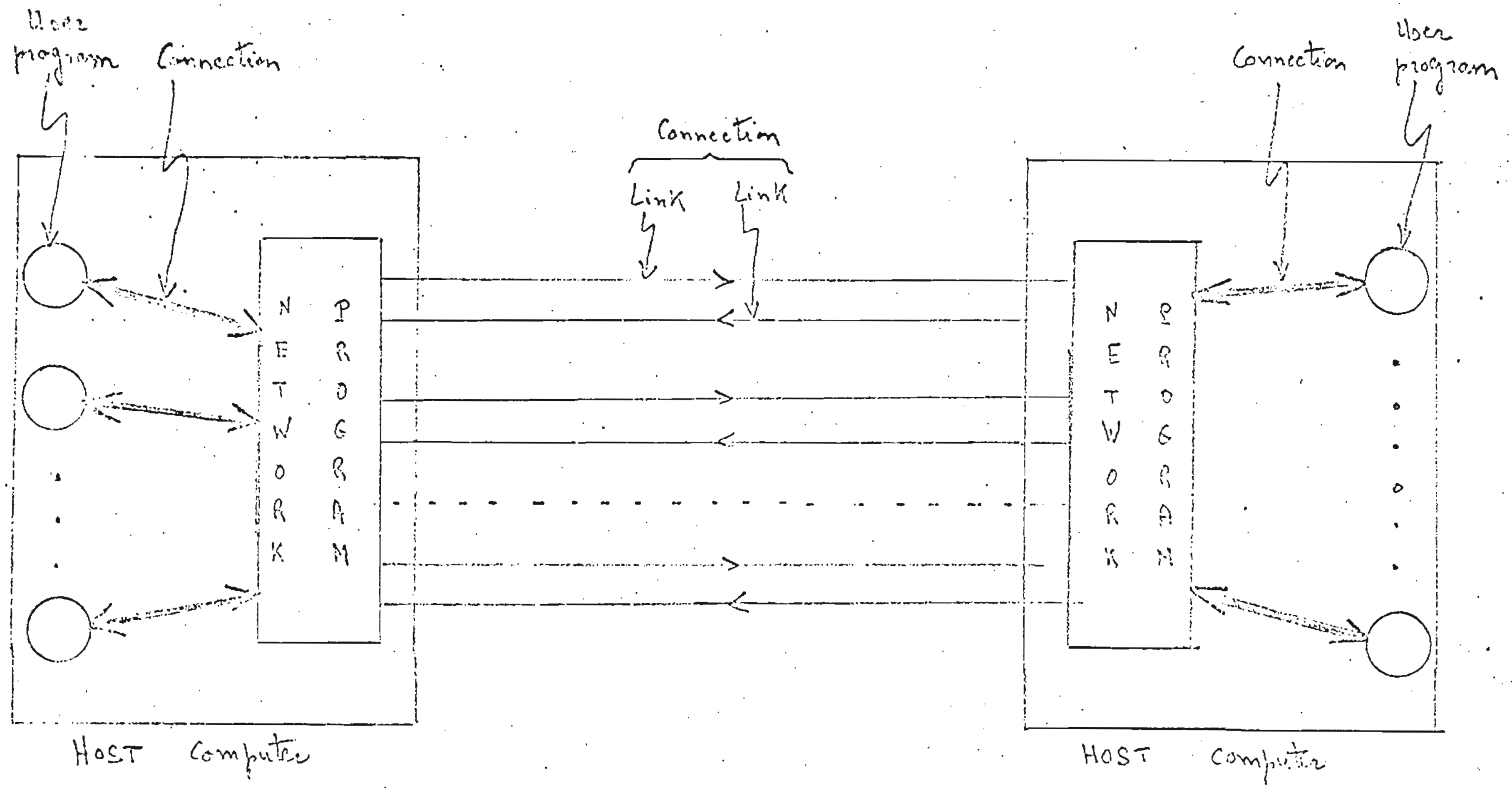
These connections also connect remote users:

An auxiliary connection:

* Is opened in parallel to a primary connection and is not unique, i.e., several auxiliary connections can be established between users.

* Is used for transmitting large volumes of data (file oriented).

* Is used either for binary or character tranmission.



Links and Connections

Figure 1

2.3 Message Structure

The HOST$_s$ communicate with each other via messages. A message may vary in length up to 8095 bits (See down below the structure). Larger transmission must therefore be broken up by HOST users into a sequence of such messages.

A message structure is identified on figure 2.

It includes the following:

(1) A leader (32 bits): Message type, Source/Destination HOST, link number. (See BBN report No. 1822, pp 13, 17)

(2) A marking (32 bits when sent by the Sigma 7) for starting a message text on a word boundary. (See BBN report No. 1822, pp. 17, 19)

(3) The message text (Max: 8015 bits for the Sigma 7). It mostly consists of user's text. However, it may represent information for use by the Network programs. (Control messages, see 2.4.2)

(4) A checksum (16 bits). Its purpose is to check, at the HOST level, the right transmission of a message. (Changes in bit pattern or packet transposition; packets are defined in BBN report No. 1763, p. 13) See down below for checksum calculation.

(5) A padding for solving word length mismatch problems. (See BBN report No. 1822, p. 17, 19.). As far as software is concerned, padding is only involved at message reception for delineating message ends. (At transmission the hardware takes care of the padding.)

Remark:

Checksum calculation:

The last 16 bits of every message sent by a HOST is a checksum. This checksum is computed on the whole message including any marking, but excluding the 32 bit leader and any padding. To compute the checksum:

1. Consider the message to be padded with zeroes to a length of 8640 bits.
2. Section the 8640 bits into six 1440-bit segments, $S_0, S_1 \ldots S_5$.
3. Section each 1440-bit segment S into 90 16-bit elements, $t_0, t_1 \ldots . t_{89}$.
4. Define a function $\oplus$, which takes two 16-bit elements as inputs and outputs a 16-bit element. This function is defined by

$$t_m \oplus t_n = t_m \oplus t_n, \text{ if } t_m + t_n < 2^{16}$$

$$t_m \oplus t_n = t_m \oplus t_n - 2^{16} + 1, \text{ if } t_m + t_n \geq 2^{16}$$

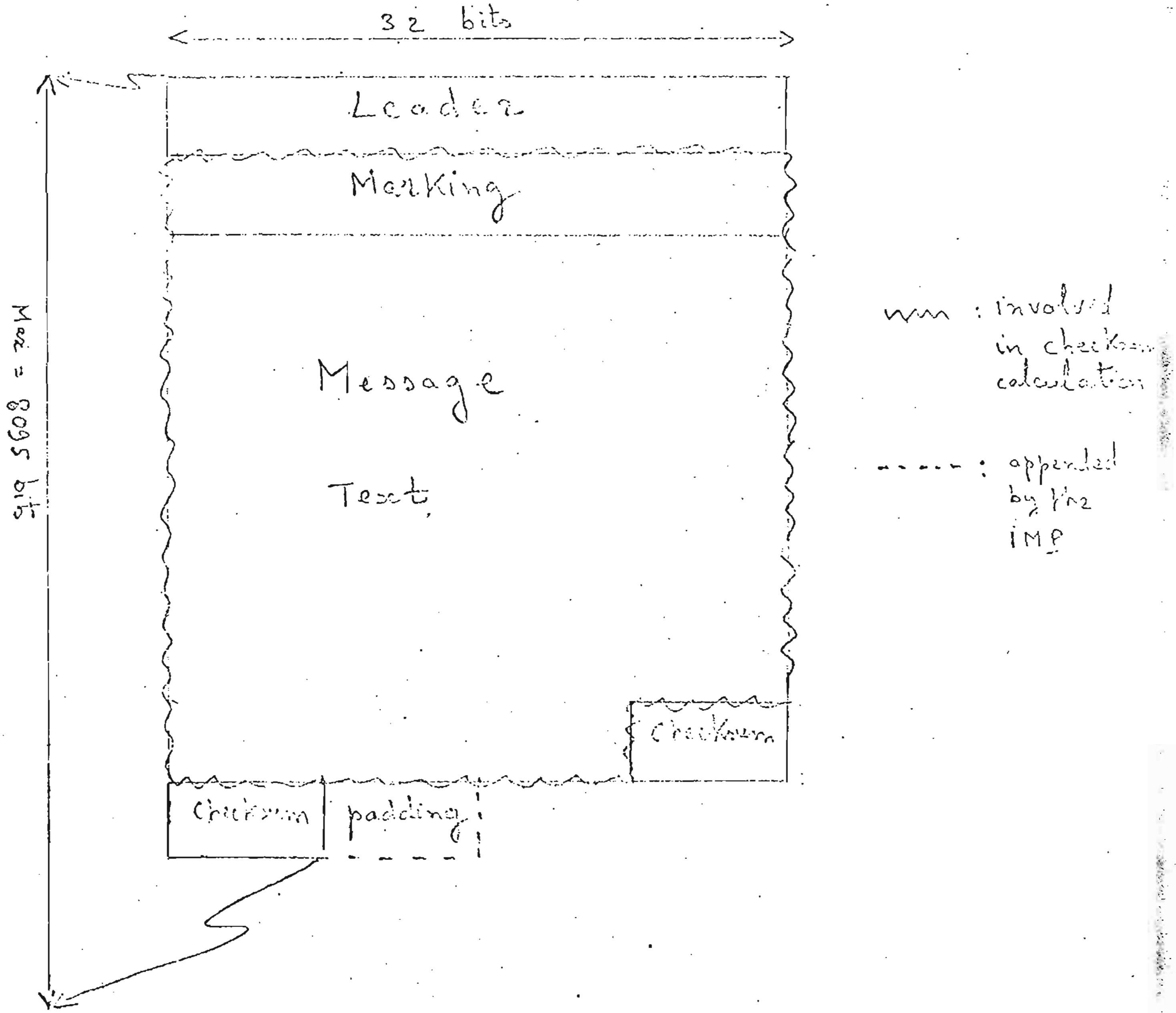
5. For each 1440-bit segment S_i compute $C_i = K(S_i)$, where

$$K(S) = t_0 \oplus t_1 \oplus \ldots\ldots t_{89}$$

6. Compute $C = C_0 \oplus C_1 \oplus C_1 \oplus C_2 \oplus C_2 \oplus C_2 \oplus C_2 \ldots\ldots \oplus C_5$

(Notice that $C_1 \oplus C_1$ is just C_1 rotated left one bit)

The number C is the checksum. The reason the C_i are rotated by i bits is to detect packet transposition.



Format of a message sent by the Sigma 7

Figure 2

2.4 User Transactions

From what has been discussed until here, the Network appears to a user as a bunch of connections. Let us now explain how one can make use of these connections.

First, we are going to describe the set of transactions that a user should be able to access for utilizing the connection facilities.

Then, we are going to explain the role of the Network program for the execution of these transactions. This will cover a HOST-HOST protocol in which control messages are exchanged between network programs.

For explanation purposes those transactions are represented, at the user level, in the form of subroutine calls and parameters. However, this does not imply at all that the implementation will closely follow this pattern. (We are more involved here with the description than the implementation aspect, see chapter 3.)

2.4.1 List of transactions

Listed below are the descriptions of subroutines that could be at user's disposal for creating/breaking connections and transmitting/receiving data over them. This set of subroutines can be considered as a kind of interface between the user level and the network program level.

(a) Open primary connection:

OPENPRIM (CONNECTID, HOSTID, BUFFADDR, [OPT])

CONNECTID: Connection identification #

HOSTID: Remote HOST identification #

BUFFADDR: Buffer address for incoming messages.

OPT: Options such as message required after successful connection establishment, "full echo" (each message is transmitted back by the remote HOST for checking purpose), etc.

Remark: [ ] means optional

(b) Open auxiliary connection

OPENAUX (CONNECTID, BUFFADDR, N, [OPT])

CONNECTID: Connection identification #, i.e., the identification of the corresponding primary connection (First a user has to open a primary connection).

BUFFADDR: Same meaning as above.

N: Number of auxiliary connections that should be opend.

OPT: Same meaning as above.

(c) Transmission over connection

TRANSM (CONNECTID, NO, BUFFADDR, N, [OPT])

CONNECTID: Connection identification #

NO: Connection #. The primary connection is always referred to as being NO = 0. An auxiliary connection number corresponds to the order in which it has been established. (The first auxiliary opened is referred to by NO = 1, the second by NO = 2, etc.)

BUFFADDR: Buffer address of the message to be transmitted.

N: Message size (byte number)

OPT: Options such as data type (characters vs. vinary), trace bit, etc.

(d) Close connection

CLOSE (CONNECTID, [N], [NO])

CONNECTID: Connection indentification #.

N: Number of connections to be closed. If omitted all connections in use by the user, included the primary link, are closed.

NO: In case of N different from zero this number indicates the auxiliary connection # to be closed.

2.4.2 HOST-HOST protocol and control messages

The HOST-HOST protocol is carried out by the Network programs. It mainly involves the execution of the previous transactions (initiated by users) and covers a HOST-HOST dialogue.

This dialogue fulfills control procedures for opening or breaking connections and consists in exchanging control messages over the control link. A control message has a structure identical to that of a regular message; it only differs from it by the text which is for use by Network programs instead of users.

Let us insist that this control procedure is completely unrelated to transmission control procedures implemented in the IMP computers. We are here at the HOST level (Network programs), and therefore control messages, that are going to be described below, are transmitted over the IMP_s like regular messages.

Consider now the previous transactions and describe for each of them which messages are exchanged over which links. Each case will be

explained by means of trivial examples.

We suppose that a HOST(x) user wants to talk to a remote HOST(y) program called URSA.

(a) <u>Open a primary connection: (OPENPRIM)</u>

The HOST (x)'s Network program, waken up (See 3.3) by a use for opening a primary connection, starts a dialogue with the HOST (y)'s Network program.

(i) HOST(x) sends the following control message:

```
HOST(x)        Control link  ------->        HOST(y)
               ENQ  PRIM  0 1 2
```

ENQ: Enquiry for connection establishment (one ASCII character)

PRIM: Connection type: primary (one special character)

0 1 2 : Outgoing link #. It is a decimal number (3 ASCII characters), e.g., link #12.

This link # has been determined by the HOST(x) Network program (See implementation: 3.3)

(ii) HOST(y) acknowledges by sending back the following control message:

```
HOST(x)   <-------  Control link               HOST(y)
            ACK ENQ PRIM 0 1 2 0 1 5
```

ACK: Positive acknowledgment (one ASCII character)

ENQ PRIM 0 1 2 : Same meaning as above. This part of the message is returned for checking purposes.

0 1 5 : Incoming link #. It follows the same pattern as the outgoing link #. This link # has been determined by the HOST(y) Network

program.

Now the connection is established; it will use links #12 and 15 for exchanging user messages. The connection is said to be in a pre-log-in state, i.e., the remote HOST(y) expects its standard log-in procedures.

(b) Transmission over primary connection: (TRANSM)

By means of TRANSM subroutines referring to the primary connection, the HOST(x) user is able to sign-in into the HOST(y) operating system and then to call for the URSA program (HOST(y) user program).

The Network programs at both ends will use the link #12 and #15 for passing along messages. These messages are standard messages whose contents serve for log in sequence.

A trivial example could be:

HOST(x) —— Prim. Link #12 ——> HOST(y)
! S I G N - I N : X X

HOST (x) <—— Prim. Link #15 —— HOST(y)
! ! R E A D Y

HOST(x) —— Prim. Link #12 ——> HOST(y)
! U R S A

(c) Open an auxiliary connection: (OPENAUXI)

In a very similar manner as (a) an auxiliary connection is established between HOST(x) and HOST(y). For so doing control messages are exchanged over the control link.

HOST(x) —— Control link ——> HOST(y)
ENQ AUX 0 2 5

HOST(x) <—— Control link —— HOST(y)
ACK ENQ AUX 0 2 5 0 2 1

Now the auxiliary connection is established, it will use links #25 and 21 for exchanging standard messages.

(d) Transmission over auxiliary connection: (TRANSM)

By means of TRANSM subroutines referring to the auxiliary connection, the users at both ends can exchange data:

HOST(x) —— Aux. Link #25 ——→ HOST(y)
X X ...... X X

HOST(x) ←—— Aux. Link #21 —— HOST(y)
X X

etc................

(e) Close connections: (CLOSE)

This is carried out in a similar manner as (a). The user calls a CLOSE subroutine and then the Network programs at both ends exchange control messages.

HOST(x) —— Control Link ——→ HOST(y)
EOT 0 0 1 0 1 2

EOT: End of transmission (one ASCII character)

0 0 1 : No. of connections to be closed (3 decimal ASCII characters)

0 1 2 : Outgoing link # to be closed.

Then HOST(y) acknowledges back as in (a).

HOST(x) ←—— Control Link —— HOST(y)
ACK EOT 0 0 1 0 1 2 0 1 5

Remark 1 - In (a), (c), and (e) HOST(y) may answer back a message including a negative acknowledgment character NAK instead of ACK. This for

many various reasons such as: wrong sequence, connection already opened, and so forth. The message could be NAK IND, where IND is an alphanumerical character indicating, in a coded form, why the previous block has been refused. Upon receiving back such acknowledgments HOST(x) will repeat its message until HOST(y) accepts it. An emergency procedure will take place if too many successive "NAK messages" occur.

Remark 2 - On each of the above illustrations (arrows) only the message text is represented. In fact, complete messages (with leader, marking, padding...) are exchanged over these links.

3. IMPLEMENTATION IN GORDO

3.1 Introduction to GORDO

GORDO is a time-sharing system implemented on SDS Sigma 7. We outline below some of the characteristics relevant to our paper.

3.1.1 GORDO file system

The file system is page oriented. It is composed of files and directories. A file consists of a heading and a number of pages which compose the body of the file. A directory consists of a number of entries that point to either files or other directories.

3.1.2 GORDO process

* A process is a program (procedures and data) plus its logical environment. In other words a process is a program which is known and controlled by the GORDO scheduler.

* A user (a job) may have several processes as different as compiler, loader, editor, application program, etc. A process is created through a system call (FORK).

* The space a process can refer to is the Virtual Space of 128k word length. A part (8k) of it is reserved for the operating system, the other part (120k) is directly accessed by the user. This later may fill or modify its part of the virtual space upon 'coupling'. (See below:

service calls) pages taken from different files. Figure 3 illustrates this coupling.

* A process can request for services by means of system calls. The system calls relevant to our paper are:

  WAKE for awaking (set active) a sleeping process

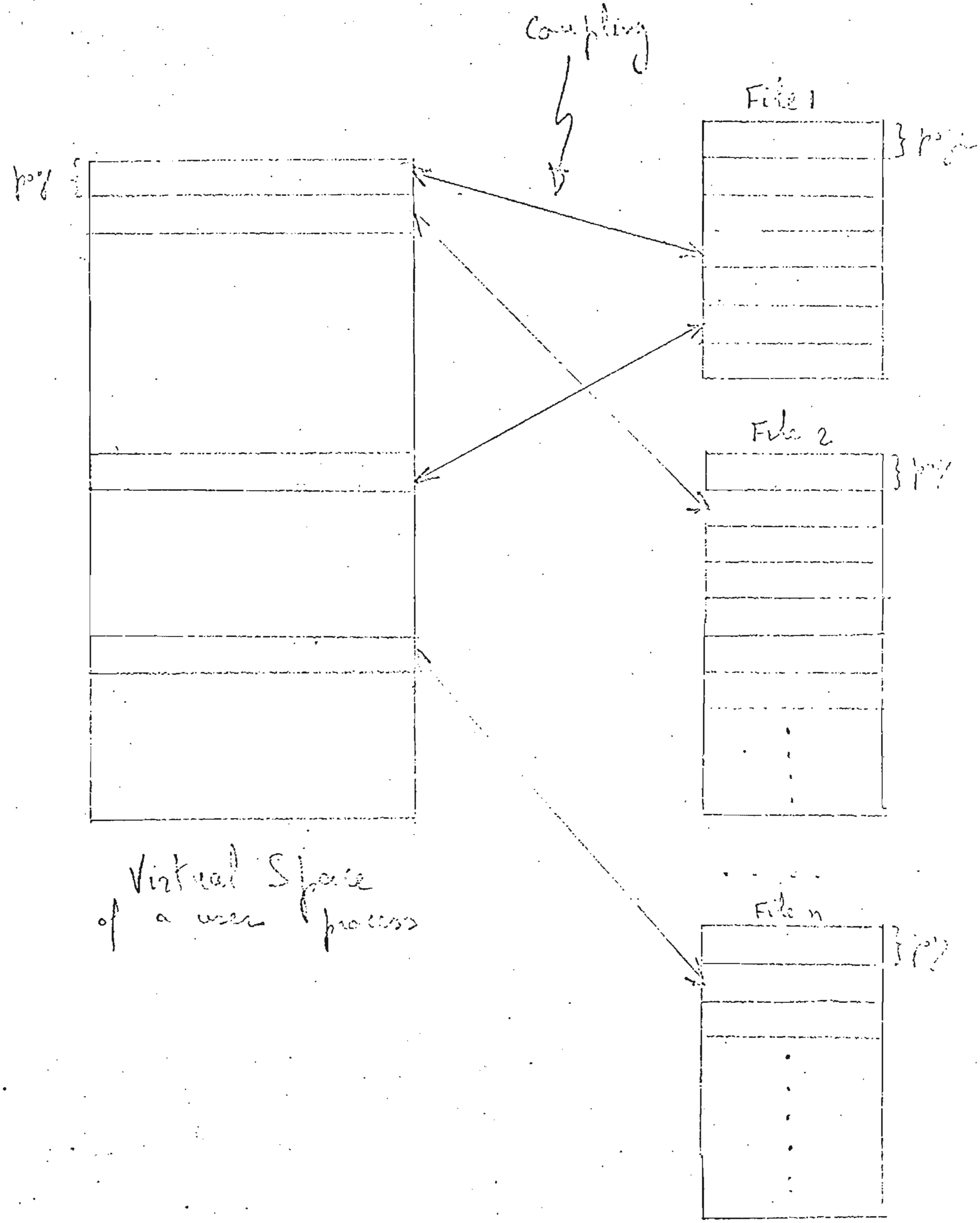
  SLEEP for putting asleep another process (or itself)

  COUPLE for coupling a page from the file space to the virtual space.

* A process ordinarily runs in slave mode. However if it is set up as an I/O process it can access privileged instructions.

* Processes can share data through files attached to "mail box" directories.

Remark: Through this note the words process and program are used interchangeably.



Virtual Space and Coupling

Figure

3.2 Software Organization Overview

Figure 4 illustrates the overall organization.

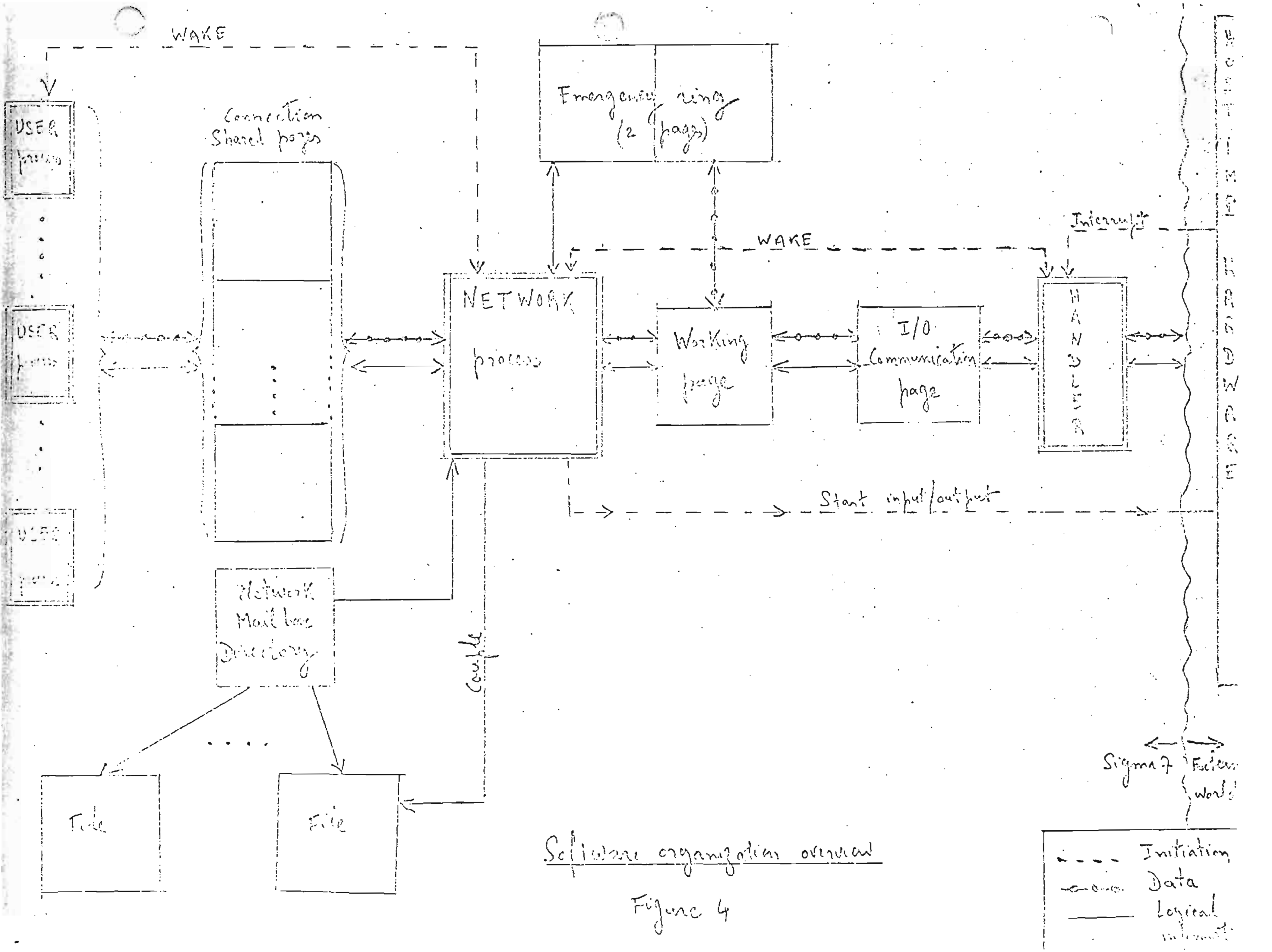
The system is based upon two main programs: the "Network" and the "Handler".

The Handler is an I/O interrupt routine closely related to the IMP-HOST hardware interface. It serves the Network process in transmitting and receiving network messages.

The Network process carries out most of the work.

Its main function is to satisfy the users' requests for opening/closing connections and transmitting/receiving network messages. For so doing,

* it establishes, identifies and breaks the links upon using the allocation tables (HOST, CONNECT, INPUT LINK; see 3.3.1.1)
* it is aware of the presence of new users upon exploring the Network mail box directory;
* it communicates with active users by means of shared pages through which messages and requests are exchanged (connection shared pages);
* it formats incoming/outgoing messages in a working page. This working page has an extension (emergency ring);
* it communicates with the Handler by means of a shared page (I/O communication page) which contains the I/O communication buffers.



Software organization overview

Figure 4

3.3 Software Description

3.3.1 Data Structures

3.3.1.1 Allocation tables: HOST, CONNECT, INPUT LINK

The Network program establishes, identifies, and breaks links and connections upon using 3 tables:

A table sorted by remote HOST #.

A table sorted by connection #

A table sorted by input link #.

(a) HOST table (See figure 5)

It is a bit table indicating the free outgoing links. It has the following characteristics:

* Location: Disc resident
* Coupling: Coupled to the Network process virtual space.
* Size: As many slots as remote $HOST_s$.
* Slot structure: As many bits as possible outgoing links to a remote HOST, i.e., 256.
* Access: Indexing. East slot is accessed through a remote HOST #.
* Specific feature: Throughout the whole table no more than 64 bits can be turned on. This figure corresponds to the maximum number of outgoing links that can be activated at one time (No matter what is the number of remote $HOST_s$).

(b) CONNECT table

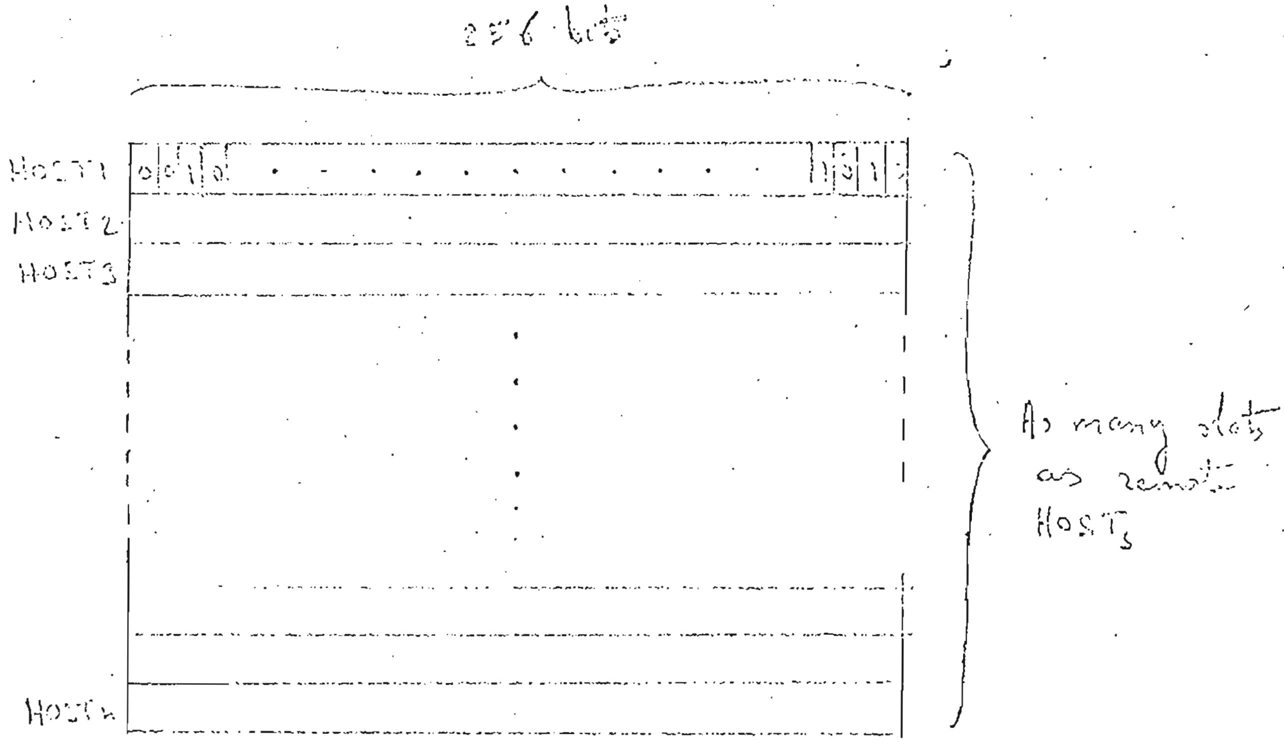
This table keeps track of all the connections' environment.

It has the following characteristics:

* Location: Disc resident
* Coupling: Coupled to the Network process virtual space
* Size: As many slots as connections in use.
* Slot structure: See figure 6. Each slot is 2 word length
* Access: Indexing. Each slot is accessed through a connection #. See 3.4 the way it is handled.
* Specific feature 1: The slot structure corresponding to a primary connection is not identical to that of an auxiliary connection (See figure 7). This because user identifications and requests are done through primary shared pages.
* Specific feature 2: This table is handled in parallel with the connection pages (See 3.3.2 (b))
* Specific feature 3: This table is mainly used for transmitting messages. (For each connection it contains the outgoing link # and remote HOST #, i.e., all the information required for transmitting a message.)

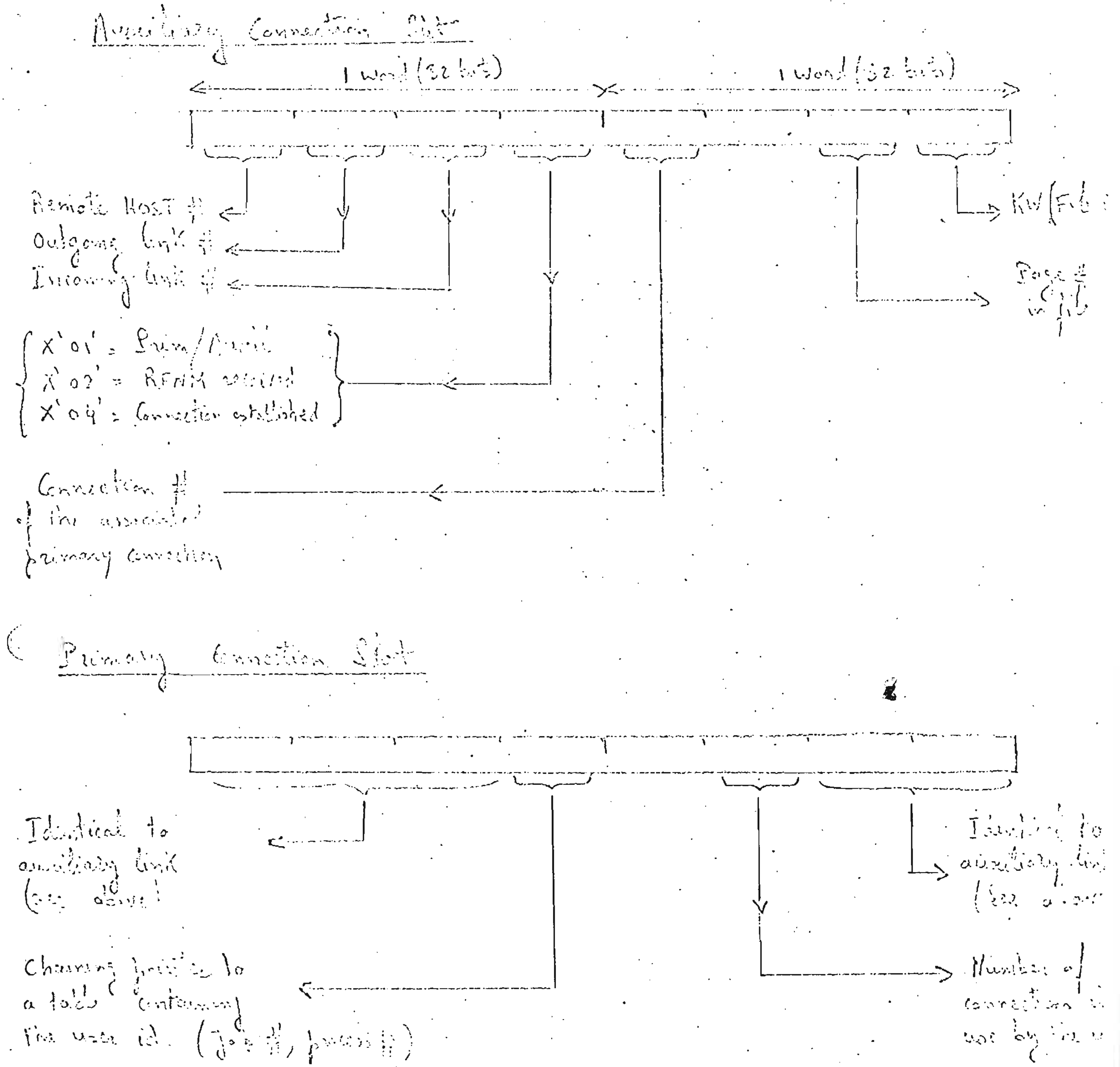
(c) INPUT LINK table

This table keeps track of all the incoming (input) links and so is closely related to the CONNECT table.



Link-face/HOST.

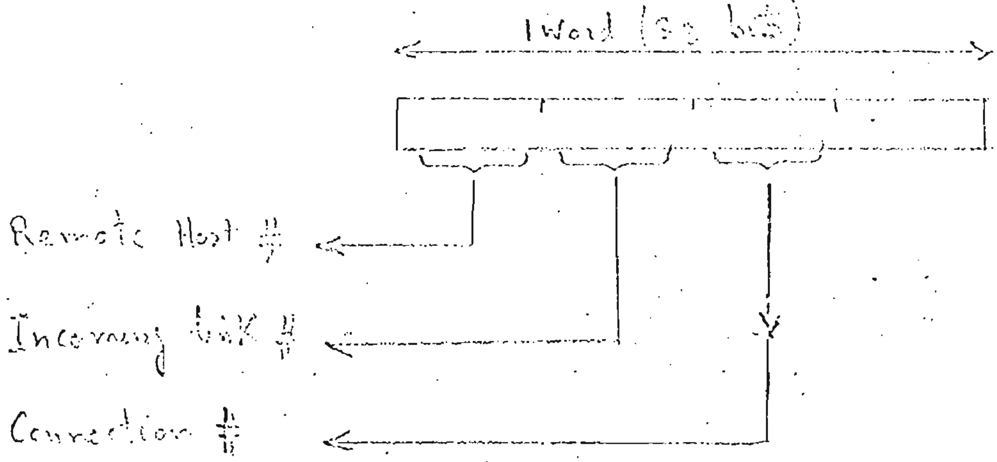
Figure 5



CONNECT table : Slot structure

(Difference between primary and auxiliary connections)

Figure 6



Remark : During a link establishment the outgoing link # is momentarily stored instead of the incoming link #.

INPUT LINK table : Slot structure

Figure 7

It has the following characteristics:

* Location: Disc resident.
* Coupling: Coupled to the Network process virtual space
* Size: As many slots as incoming links, i.e., as connections
* Slot structure: See figure 7. Each slot is 1 word length
* Access: Hashing. The hashed key value is mainly based upon the incoming link # and the remote HOST #.
* Specific feature 1: This table is also used for momentarily memorizing the connection number while establishing the next connection. See 3.4 the way it is handled.
* Specific feature 2: This table is primarily used upon receiving messages. (For each incoming link it contains the corresponding connection #, i.e., indirectly the user identification to which the message should be passed along)

3.3.1.2 Buffer pages

All the pages that are now to be described contain two buffers (input and output). These buffers are used for either passing along or processing messages.

The size of each of these buffers should at least be equal to that of a message, i.e., 8095 bits. We have chosen a buffer size of 253 words (8096 bits) so that both of the buffers are included within one page (512 words). The 6 remaining words of the page are generally used for control.

A typical buffer page structure is identified on figure 8.

(a) I/O communication page

See figure 9.

This I/O communication page is used as an interface between the Handler and the Network program.

In the buffers of this page the messages are assembled (input) or de-assembled (output) word by word by the Handler, e.g., a "ready to go" message, sorted by the Network program in the output buffer, is shipped out word by word by the Handler.

Main characteristics:

* Location: Resident in core: Locked page
* Coupling: Coupled to the Network process virtual space
* Content:
  * Input buffer (253 words) for incoming messages
    Output buffer (253 words) for outgoing messages
  * Input control zone (6 half words)
  * Output control zone (6 half words)
* Structure: See figure 9.
* Specific feature:
  * The input buffer is filled by the Handler (read from hardware) and emptied by the Network program
  * Vice versa for the output buffer

(b) Connection shared pages (User-Network shared zone)

General features:

* There are as many shared pages as connections.

* These pages shared between the network and the user processes constitute a communication zone for (1) passing the messages back and forth, and (2) exchanging control information, e.g., a request for establishing new connections.

Main characteristics:

* Location: Disc resident
* Coupling: Coupled to both a user process virtual space and the network process virtual space.
* Content: - Input buffer (253 words) for incoming messages
  - Output buffer (253 words) for outgoing messages
  - Input control zone (6 half words)
  - Output control zone (6 half words)
* Structure: See figure 10.
* Specific feature 1: - The input buffer is filled by the Network and emptied by the user.
  - Vice versa for the output buffer.
* Specific feature 2: The control zone corresponding to a primary connection shared page differs from that of an auxiliary connection. This because it is via a "primary connection control zone" that auxiliary connection establishment requests are transmitted to the Network process.

(c) Working page

General feature:

* This page allows the Network and the Handler programs to work

independently on different messages and so contributes to an overlapping. For instance, when the Handler is busy transmitting a message to the hardware, the Network program can format (leader, marking, etc.) the reset message to be shipped out, so that it can reinitiate the Handler as soon as it is free.

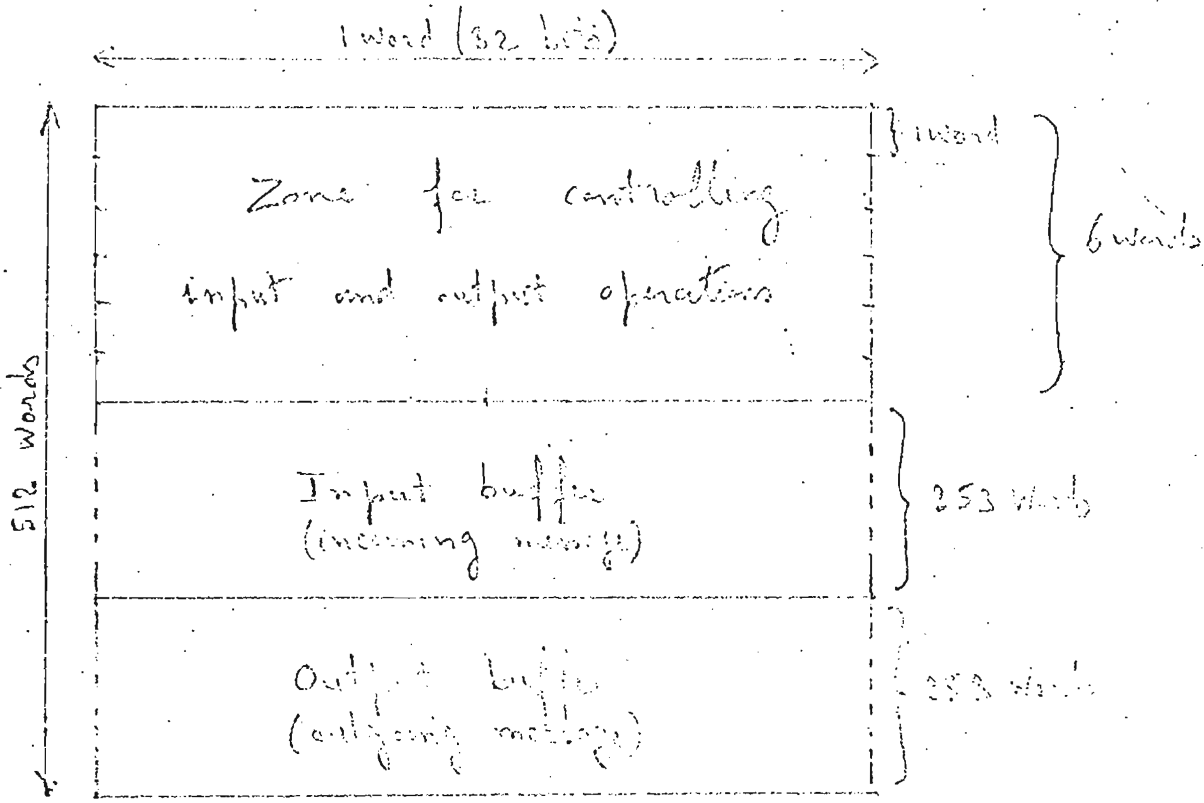
Main characteristics:

* Location: Disc resident
* Coupling: Coupled to the Network process virtual space
* Content: - Input buffer (253 words) for incoming messages
  - Output buffer (253 words) for outgoing messages

Remark:

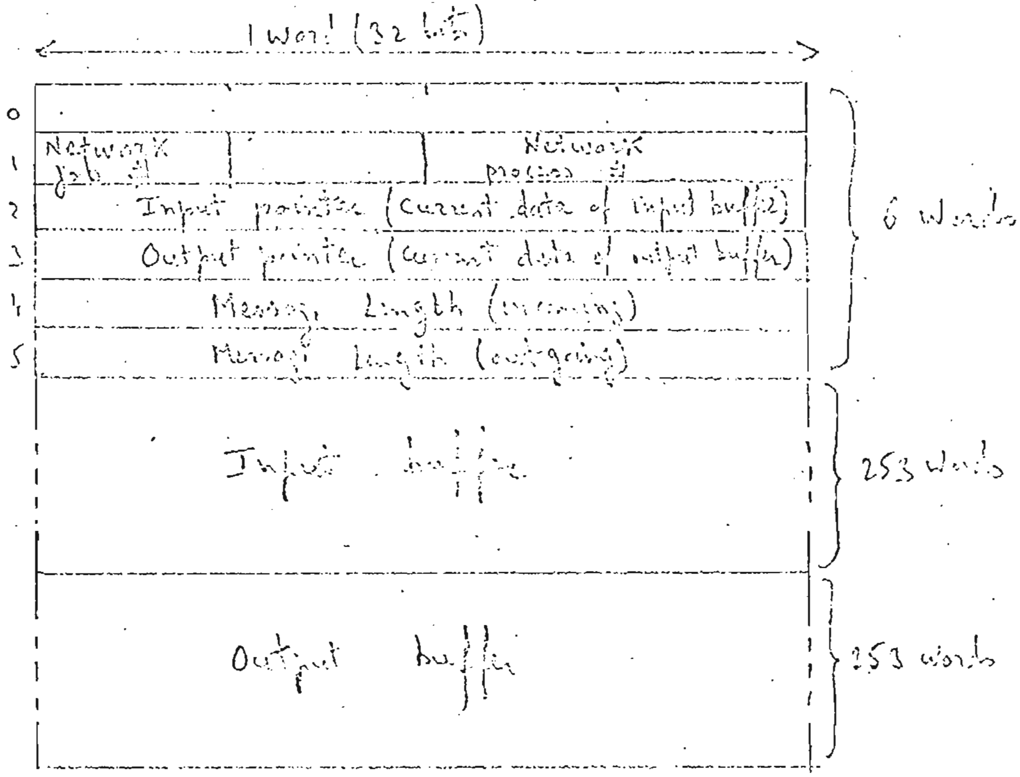
During reception it may happen that a user program is not ready to accept a new message. In that case, to avoid clogging up the system, the Network stores momentarily the incoming message in one of the buffer of the emergency ring. (If this ring is full a help routine will be invoked.)

During emission all operations are synchronized with the $RFNM_s$, therefore such procedures need not be provided. (The Network program allows a user to re-emit only when having received the RFNM of the previous transmitted message.)



Typical buffer page

Figure



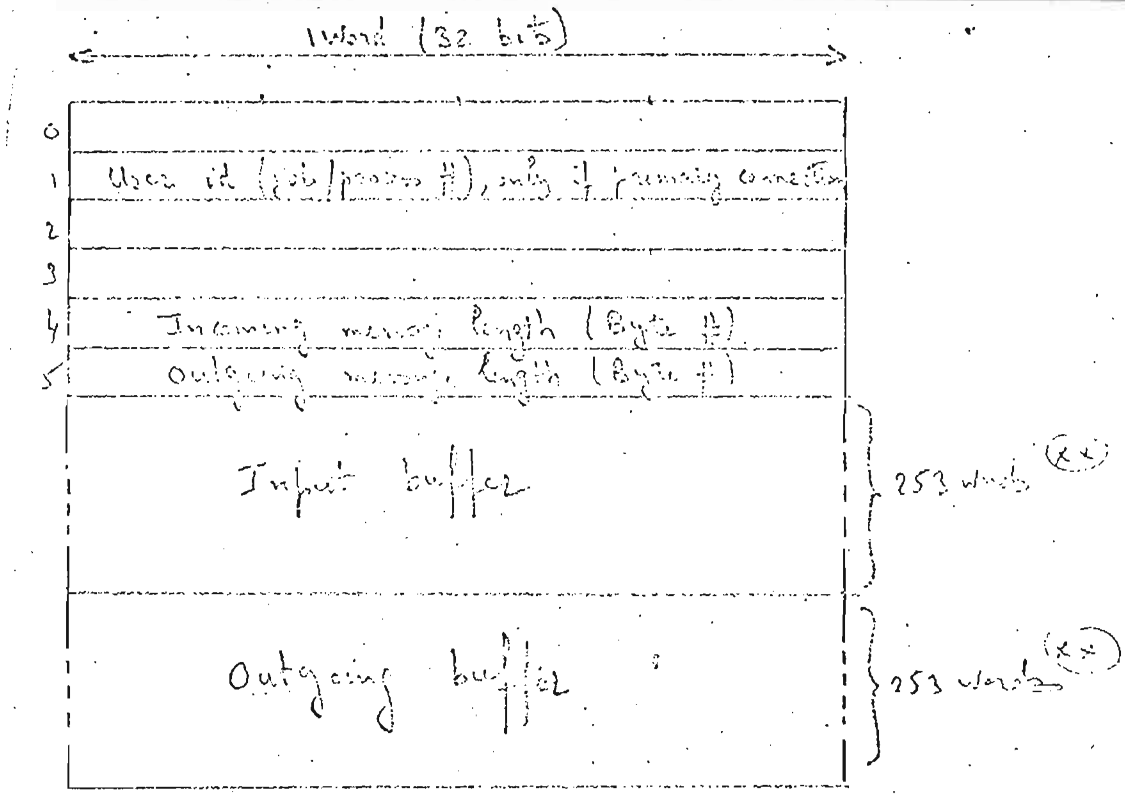
Word 0 structure:

X'80000000' = "Thrurecv" bit
X'00008000' = "Thrusend" bit } Dialogue with Network proc.
X'10000000' = "OK recv" bit
X'00001000' = "OK send" bit } Dialogue with HOST-IMP hardware.

I/O communication page structure

(Locked page)

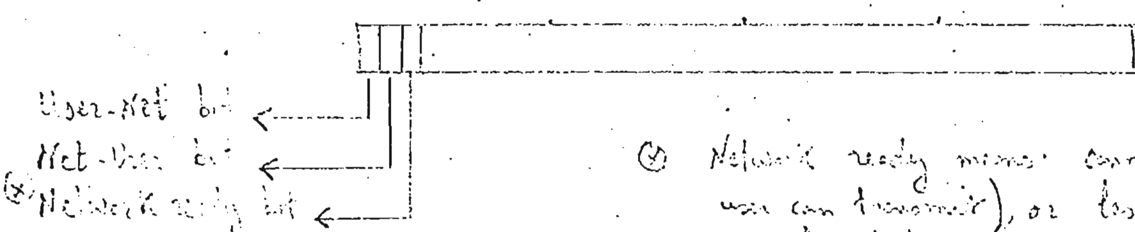
Figure 9



(xx) In fact the user should not use the 253 words (Room is needed for leaders, marking, etc)

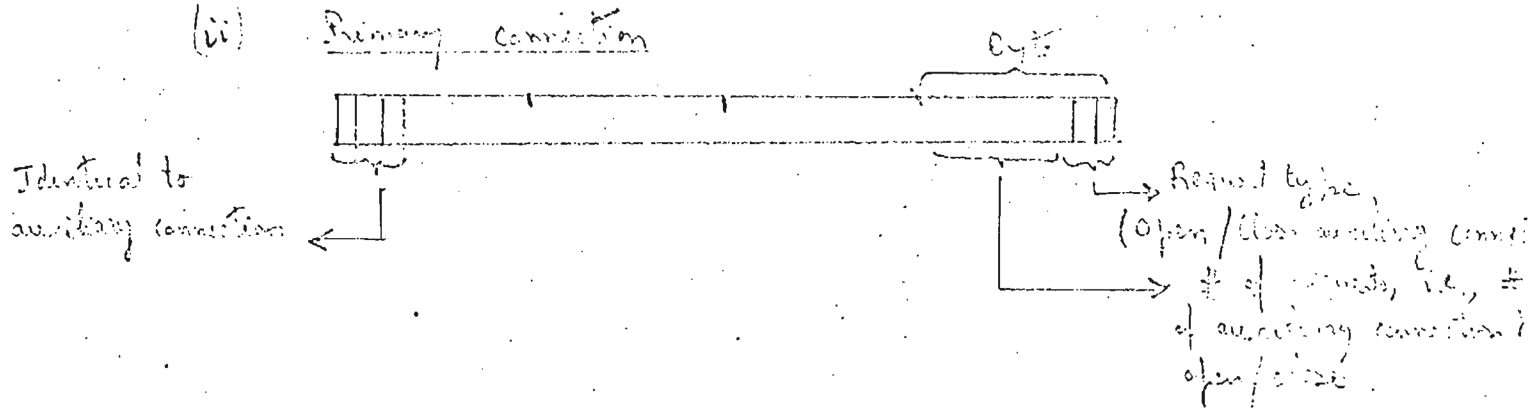
Word 0 structure

(i) Auxiliary connection



(x) Network ready means connection is established (the user can transmit), or last transmission has been acknowledged by the IMP: RFNM received.

(ii) Primary connection



Connection shared page structure

Figure 10

3.3.2 Programs

3.3.2.1 Handler program

General features:

It is an I/O interrupt routine which drives the IMP/HOST hardware interface in order to transmit or receive messages. Transmission and reception are carried out in a full duplex mode.

Main characteristics:

* Location: Core resident. The Handler is in the same memory zone as the operating system and can be considered as part of it.
* Initiation: By the IMP-HOST hardware interrupt. This interrupt is triggered either:
    * during transmission when a message word is completely sent to the IMP
    * during reception when a message word has been completely received from the IMP
    * during idle time when the hardware received either a 'start input' or 'start output' order from the Sigma 7 CPU. Those orders are issued by the Network program for provoking interrupts back (consequently for indirectly initiating the Handler).
* Main functions: * Empties the output buffer upon transmitting its content (outgoing message to the IMP. This operation is carried out word by word (32 bits) and makes use of "Write" orders for

driving the HOST-IMP hardware.

* Fills the input buffer with data received from HOST-IMP hardware (incoming message). This operation is also carried out word by word and makes use of "Read" orders for driving the HOST-IMP hardware.

* Wakes up the Network program when any of the previous operations is complete.

3.3.2.2 Network program

General features:

This program serves the user for opening/closing connections and transmitting/receiving messages. It uses the Handler as an aid for interfacing with the hardware.

For the GORDO point of view it is a regular process and treated as such.

Main characteristics:

* Location: Disc resident. More precisely it is on disc when asleep and called in core when awakened by a program.

* Initiation: It is initiated through 'WAKE' service calls issued either by a user process or by the Handler.

* Main functions: * Establishes/deletes outgoing connections upon users' requests. For so doing it sends control messages (see 2.4.2) to remote $HOST_s$ in order to get links established/released; it then notifies back the users.

* Insures the processing of incoming control messages (transmitted over control links), e.g.,

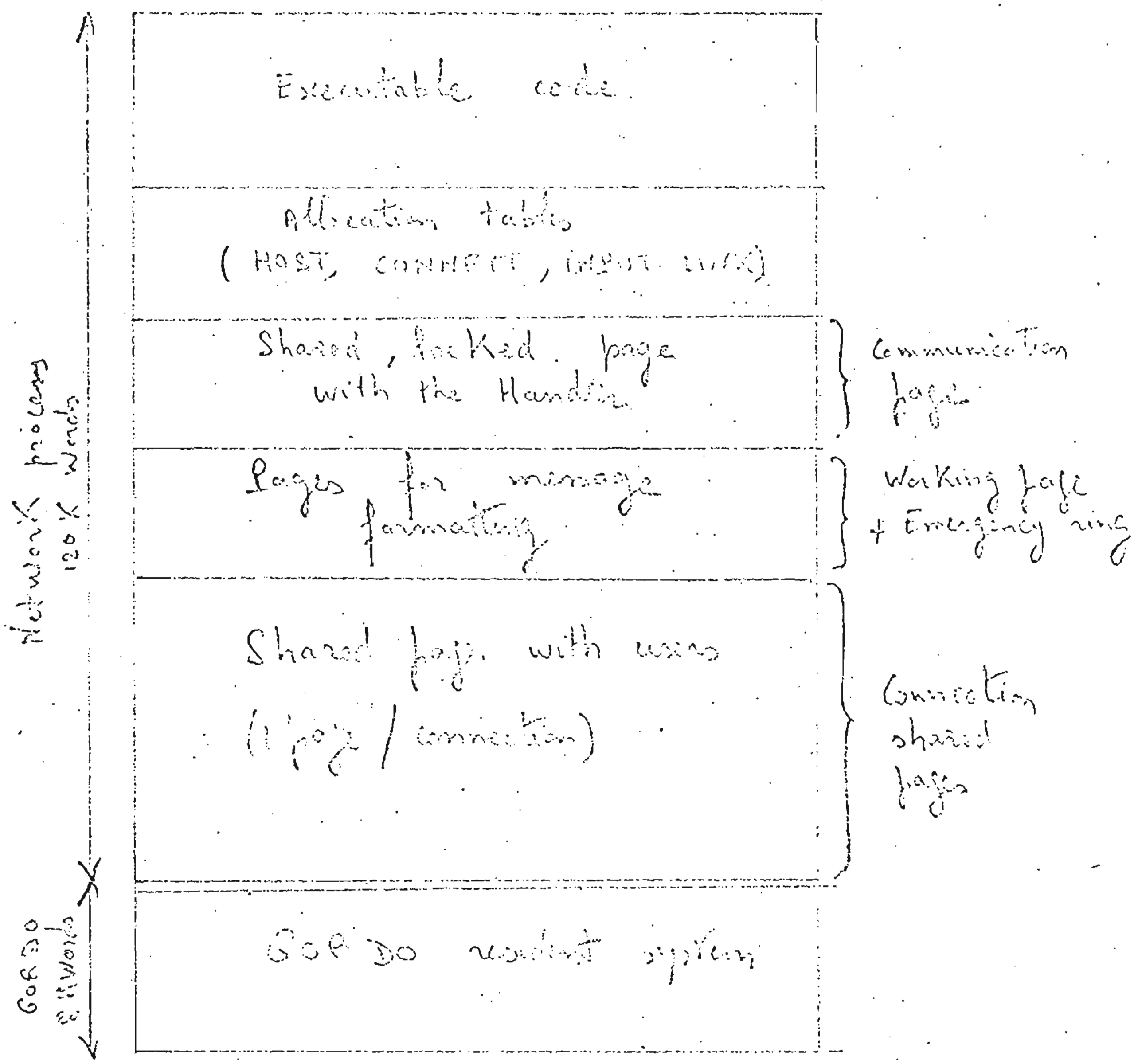
for contributing to establishments/deletions of connections (those requested by remote HOSTS).

* Prepares transmission of outgoing messages. It picks up text messages from shared pages (the messages are stored there by users), formats them (adds leader, marking, checksum..), and passes them along to the Handler for transmission.

* Insures delivery of incoming messages. It is the opposite of the above operation. The users to which the messages should be delivered are identified through the leaders.

* Virtual space configuration: See figure 11.

* Specific feature: It is integrated as an I/O process, so that it can access privileged instruction (RD/WD for indirectly initiating the Handler).



Network process virtual space

Figure 11

3.4 Software Procedures

The detailed sofrware procedures are given on the flowcharts attached with Appendix A.

However, to get a quick understanding of the implementation we list below some typical software procedures.

3.4.1 Description of some typical sequences

Consider some of the transactions at user's disposal (See 2.4) and point out the basic software procedures they imply. For each case we will delineate (i) what the user program does and (ii) what the Network program does.

(a) Open a primary link (See also 2.4.2)

(i) What the user program does[1]:

* it stores in the Network mail box directory the name of a file, e.g., DATA;
* it couples the first page of this file to its virtual space;
* it stores information in this page (its job /process #, the remote HOST #, e.g., (i));
* it wakes up the Network process;
* it goes to sleep.

(ii) What the Network program does:

* it explores the Network mail box directory and accesses the file DATA;

* it couples the first page of this file to its virtual space (Shared Zone, see 3.3.1.2). Suppose this paget be the kth in the shared zone; k is the internal connection #;
* it explores the ith slot of thw HOST table (See 3.3.1.1 (a)) and selects the first bit = 0, e.g., the αth bit; α corresponds to the outgoing link #;
* it stores information (job/process #, remote HOST # (i), outgoing link # (α)) in the kth slot of the CONNECT table (See 3.3.1.2).
* it momentarily stores the connection # (k) in the INPUT LINK table. This is carried out upon creating an entry in this table (Hashing the key value: "outgoing link # (α) + remote HOST # (i) + outgoing flag".);
* it prepares the message text ENQ PRIM 0 0 α and formats a complete message in adding leader, marking, checksum, etc.;
* it checks the Handler state (bit in I/O locked page). <u>If</u> the Handler is free, it stores the 'ready to go' control message in the output buffer of the I/O locked page, initiates the Handler, and goes to sleep. <u>Else</u> it goes to sleep.

After a while the Handler wakes up the Network process because it has received a complete message. We suppose this message be the control message sent by the remote HOST for acknowledging the establishment of the connection. The message text should be:

ACK ENQ PRIM 0 0 α 0 0 β

where β is the incoming link #. (See 2.4.2)

Let's see now what the Network program does when receiving the above control message:

* it retrieves the connection # previously stored in the INPUT LINK table upon re-hashing the same key value (See above). Also it deletes this entry;
* it creates an entry in the INPUT LINK table for the incoming link. For so doing it hashes the key value: "incoming link # (β) + remote HOST # (i) + incoming flag". In this entry it stores the HOST # (i), the incoming link # (β), and connection # (k);
* it updates the kth slot of the CONNECT table in storing the incoming link # (β);
* it turns on the 'net-user' bit in k^{th} shared page (page corresponding to the primary connection that has just been opened) and wakes up the user process;
* it goes to sleep.

(b) Transmit a message over primary link

(i) What the user program does [1].

* it stores the message text in the output buffer of the primary connection shared page (See 3.3.1.2);
* it turns on the 'user-net' bit of this page and wakes up the Network process;
* it goes to sleep.

(ii) What the Network program does:

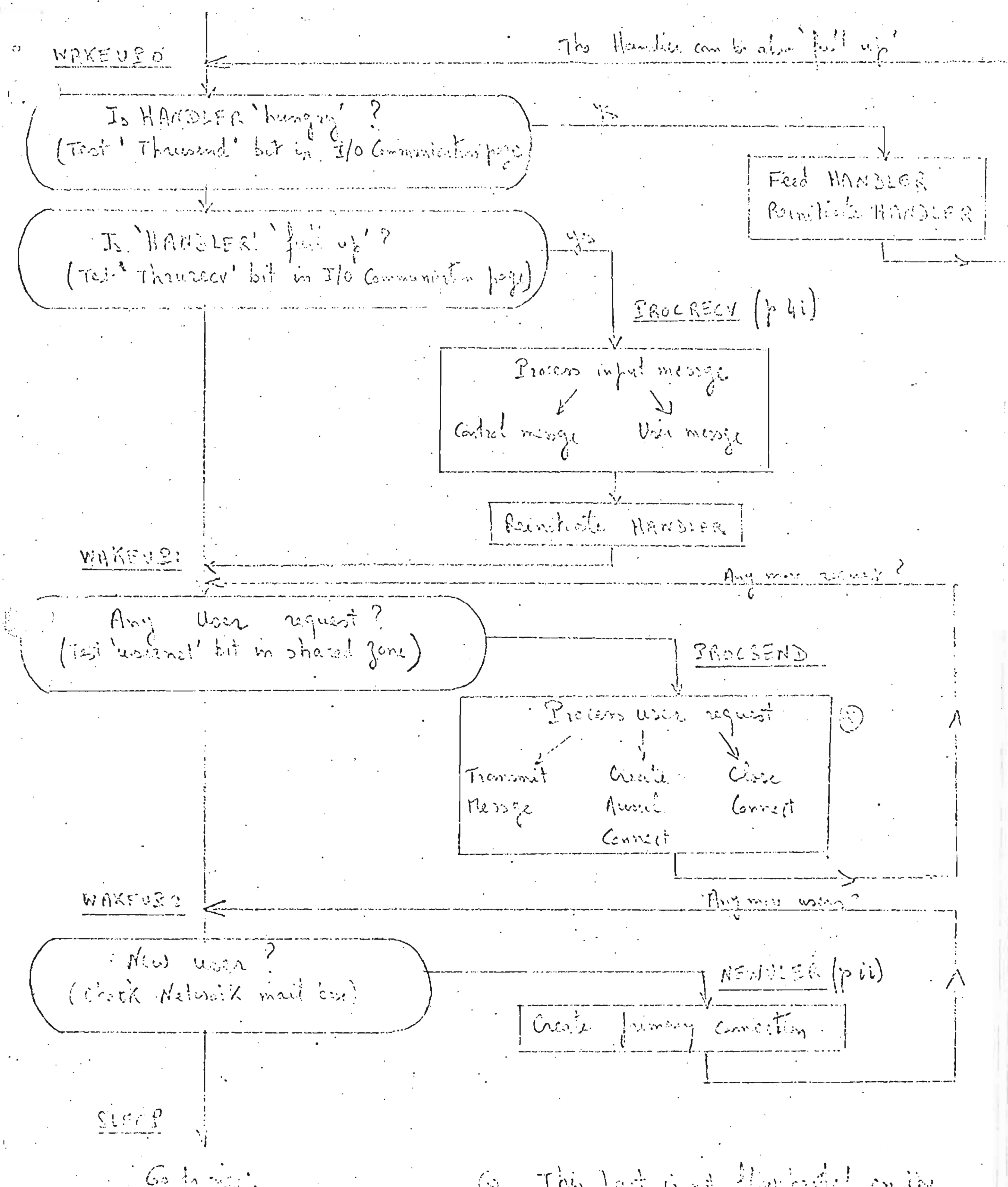
* it looks for user request, i.e., it explores in sequence the connection shared pages and selects the one that has its'user-net' bit turned on. Suppose k be the selected page # on the shared list, K is the connection #;
* it determines the request type in testing the "request bits' of the shared page k. It finds out that it is a request for transmitting a message.
* it takes the message text from the output buffer of the shared page k, formats it into a complete message and transmits to the Handler in a very similar way as above (See Open a primary link).
* it goes to sleep.

[1] Remark: In a first phase the user will directly write the network functions in his program. Later on subroutines will be put at user's disposal. These subroutines will be very close to those described in 2.4.

APPENDIX A

Flowcharts

NETWORK PROCESS Organization



(*) This part is not flowcharted on the following pages. It is very similar to the other parts

NEWUSER

↓

Compile a new shared page

- Take the first page (will be 'primary' shared page) of the file referred to in the Network mail box & copy it to the Network virtual space (Shared list).
- Assume (h) be { the page # index / connection # } on the shared page list

↓

Get information stored in this page by the user

- Remote HOST # (i)
- Job/Process #

↓

Select an outgoing link #

- Explore i-th entry of HOST table & select the first (α) bit on; outgoing link # = (α)

↓

Update CONNECT table

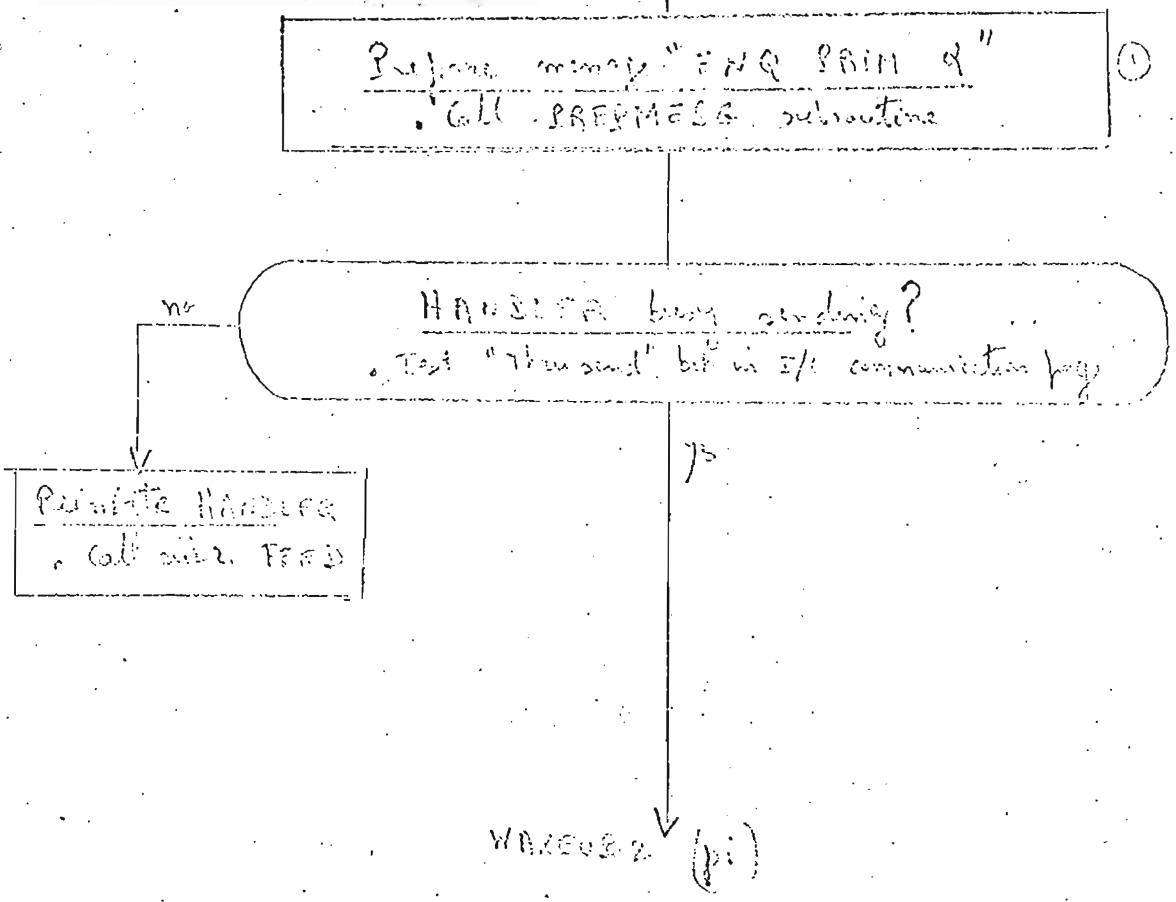
- Access h-th entry & store:
  - Remote HOST #
  - Job/Process #
  - XW (File id), Page # in file (=1)
  - Outgoing link # (α)

↓

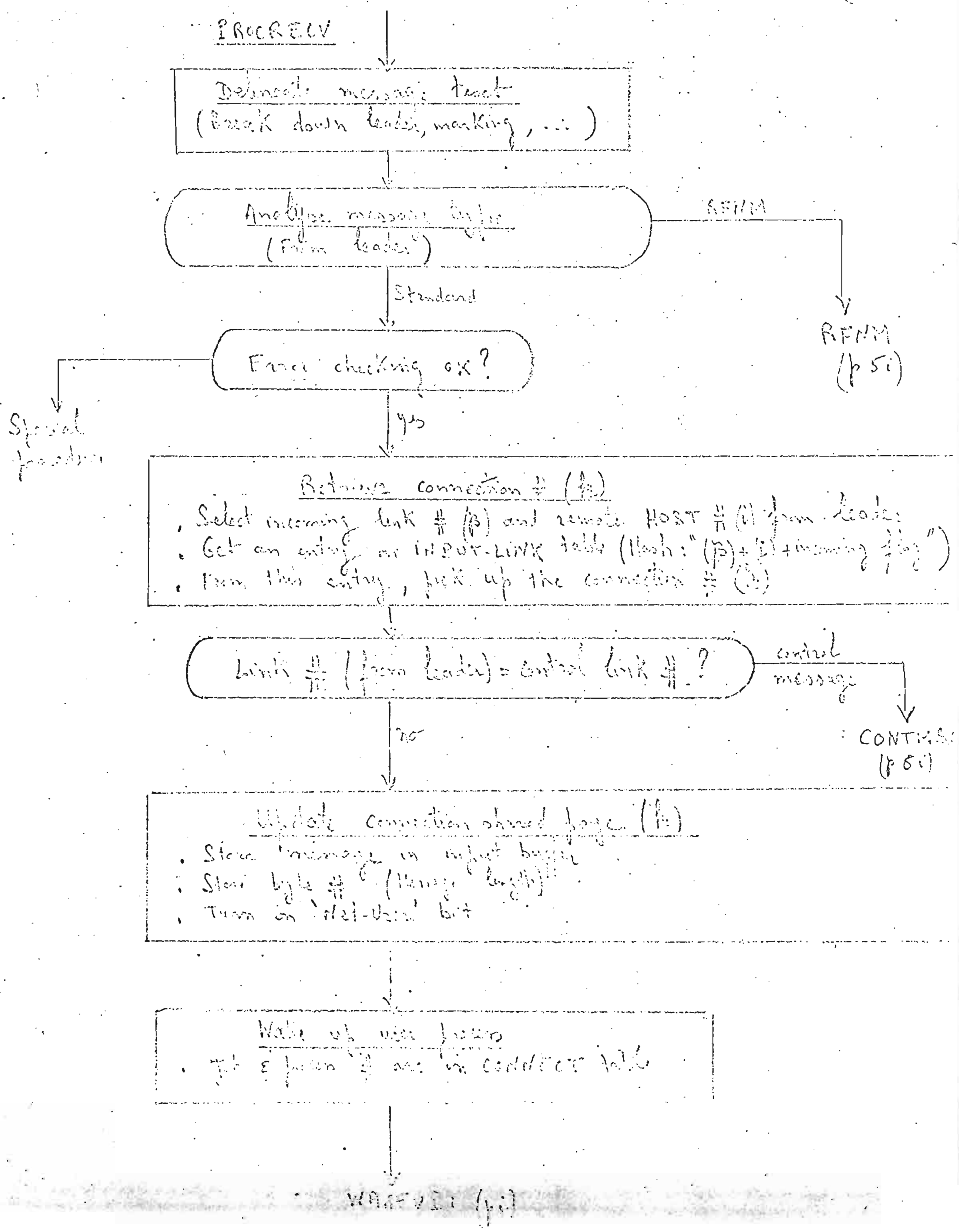
Protect CONNECT #

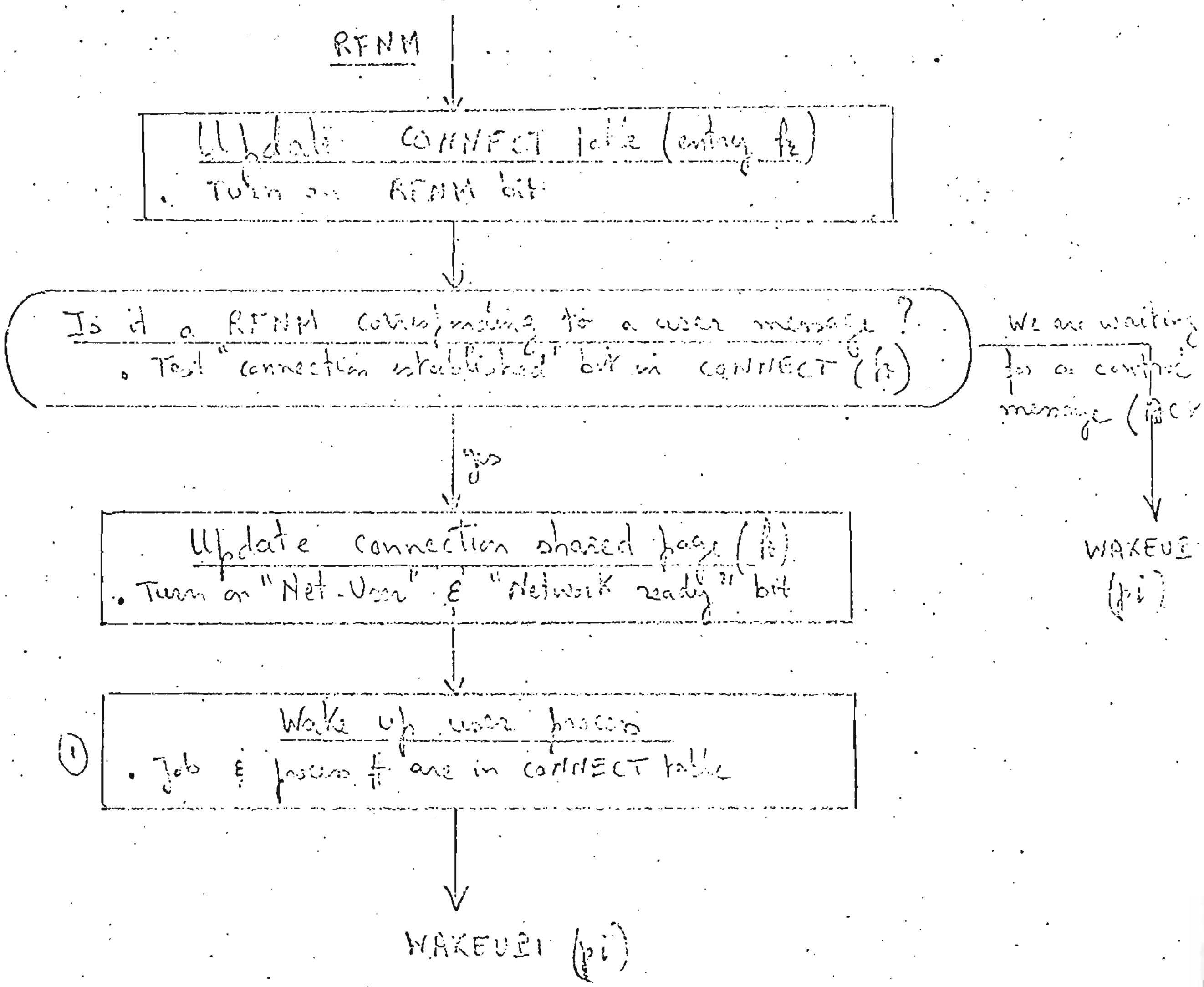
- Store momentarily the connection # (h) in the INPUT LINK table. The [illegible] as an entry is accessed upon hashing the key value: "outgoing link # (α) + remote HOST # (i) + outgoing [illegible]"

↓

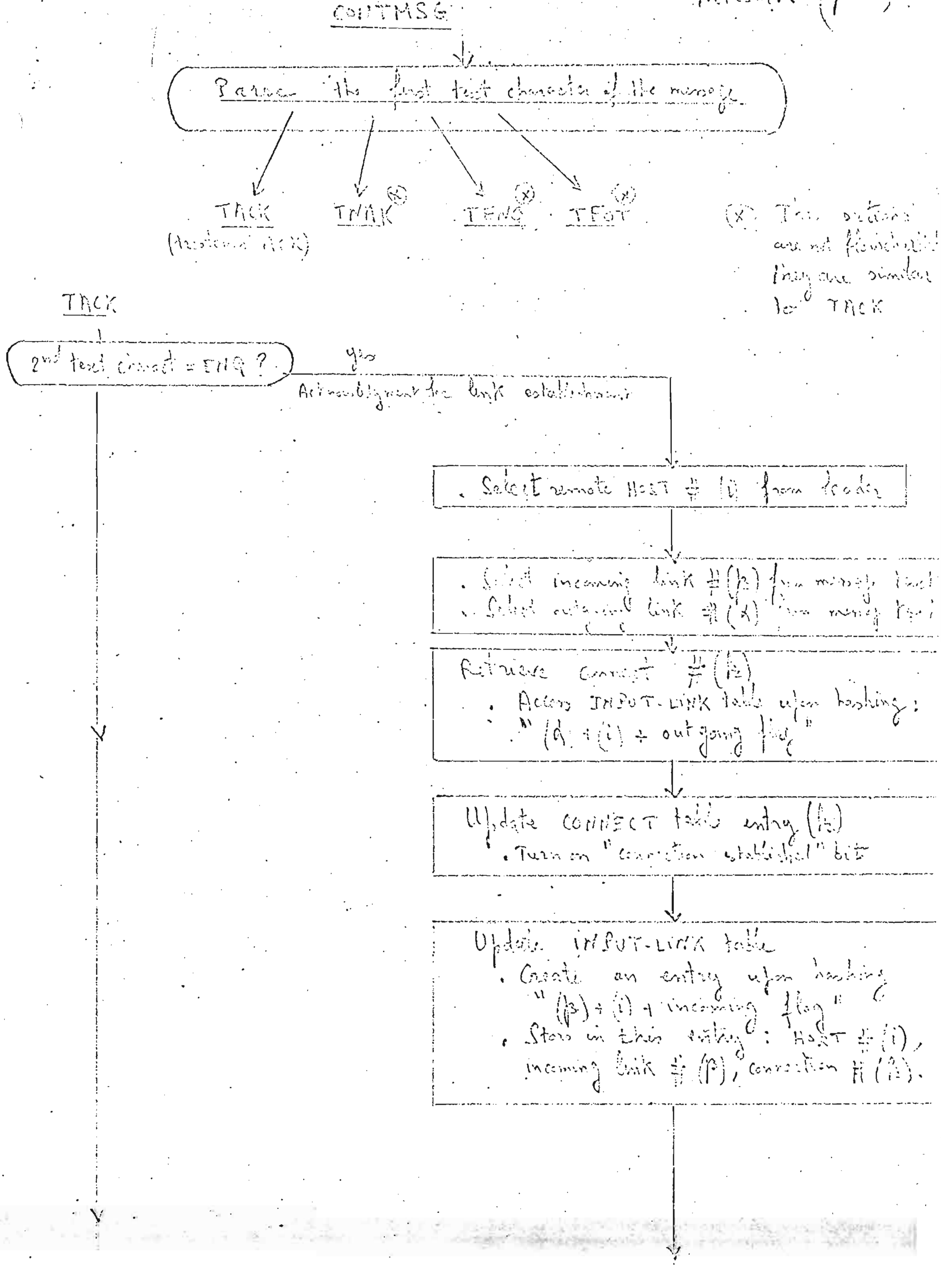


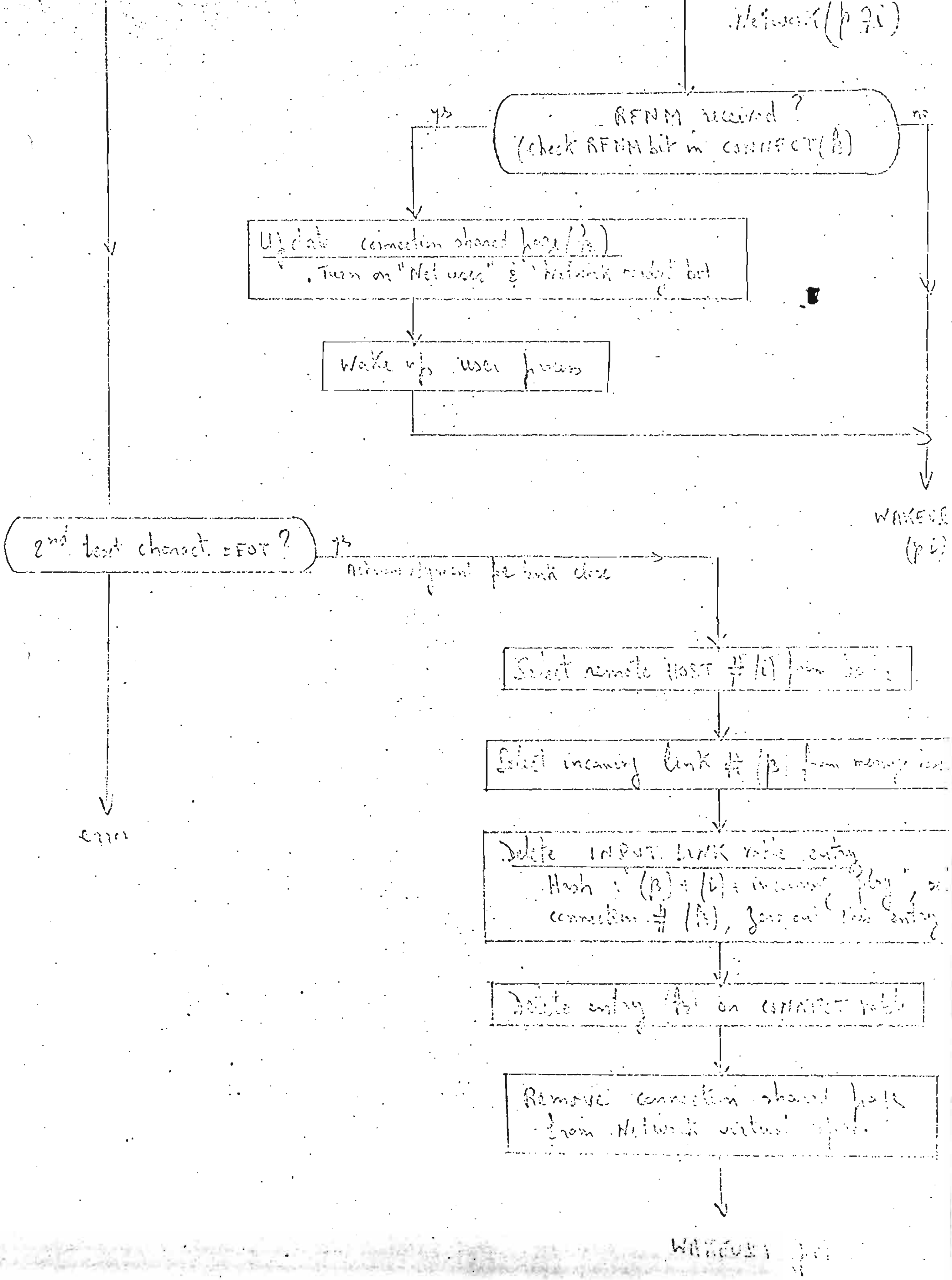
(1) PREPMESG subroutine formats the message to be shipped out (framing marking, checksum, etc)


PROCRECV
Delineate message text
(Break down leader, marking, ...)
Analyze message type
(From leader)
RFNM
Standard
RFNM
(p 5i)
Error checking OK?
Special procedure
yes
Retrieve connection #
. Select incoming link # (β) and remote HOST # from leader
. Get an entry on INPUT-LINK table (Hash: "(β) + incoming")
. From this entry, pick up the connection #
Link # (from leader) = control link #?
control message
CONTMSG
(p 6i)
no
Update connection shared page
. Store message in input buffer
. Store byte # (message length)
. Turn on 'Net-User' bit
Wake up user process
. Get process # in CONNECT table
WAKEUP (p i)

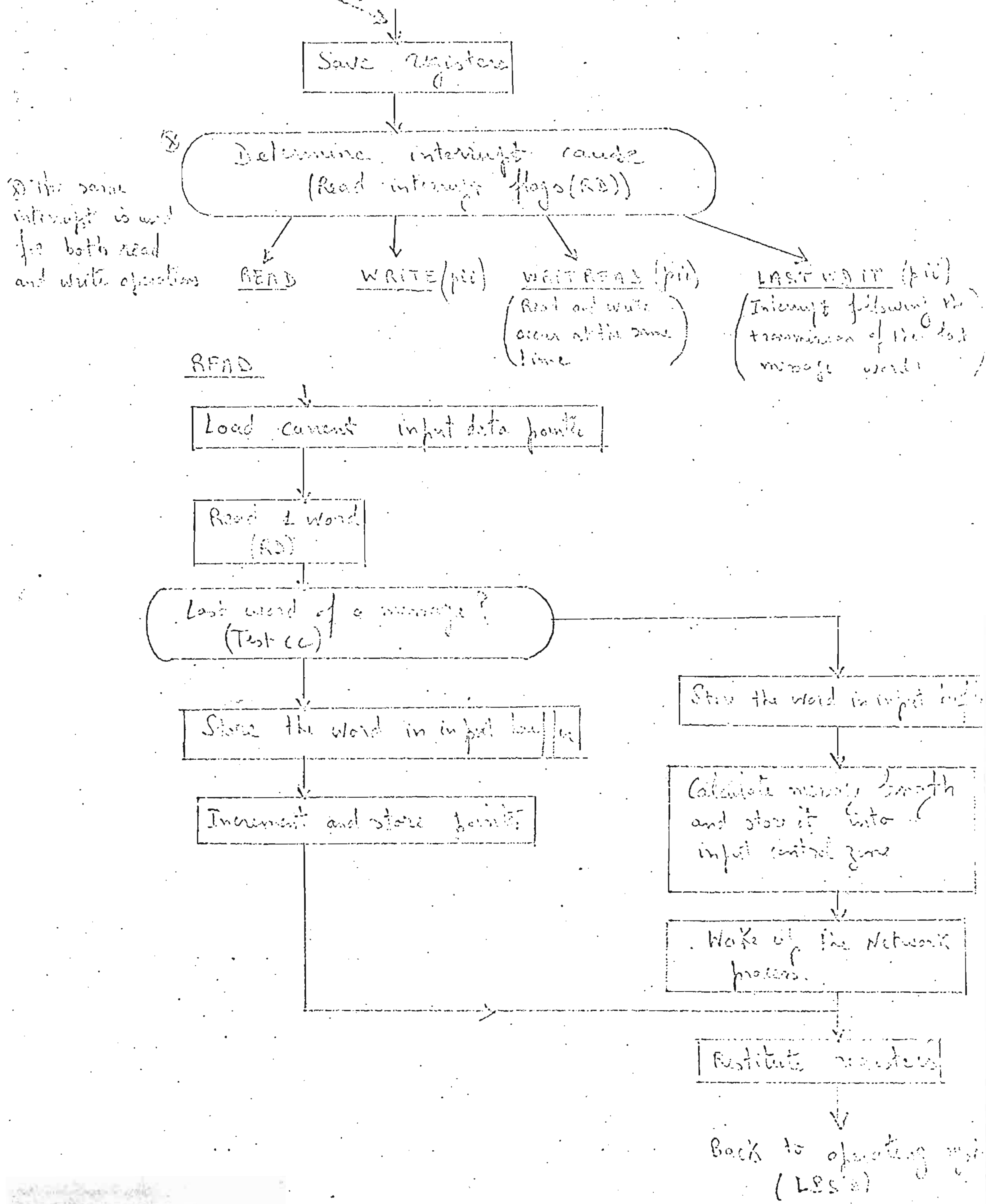


(1) The user can start on said transmission

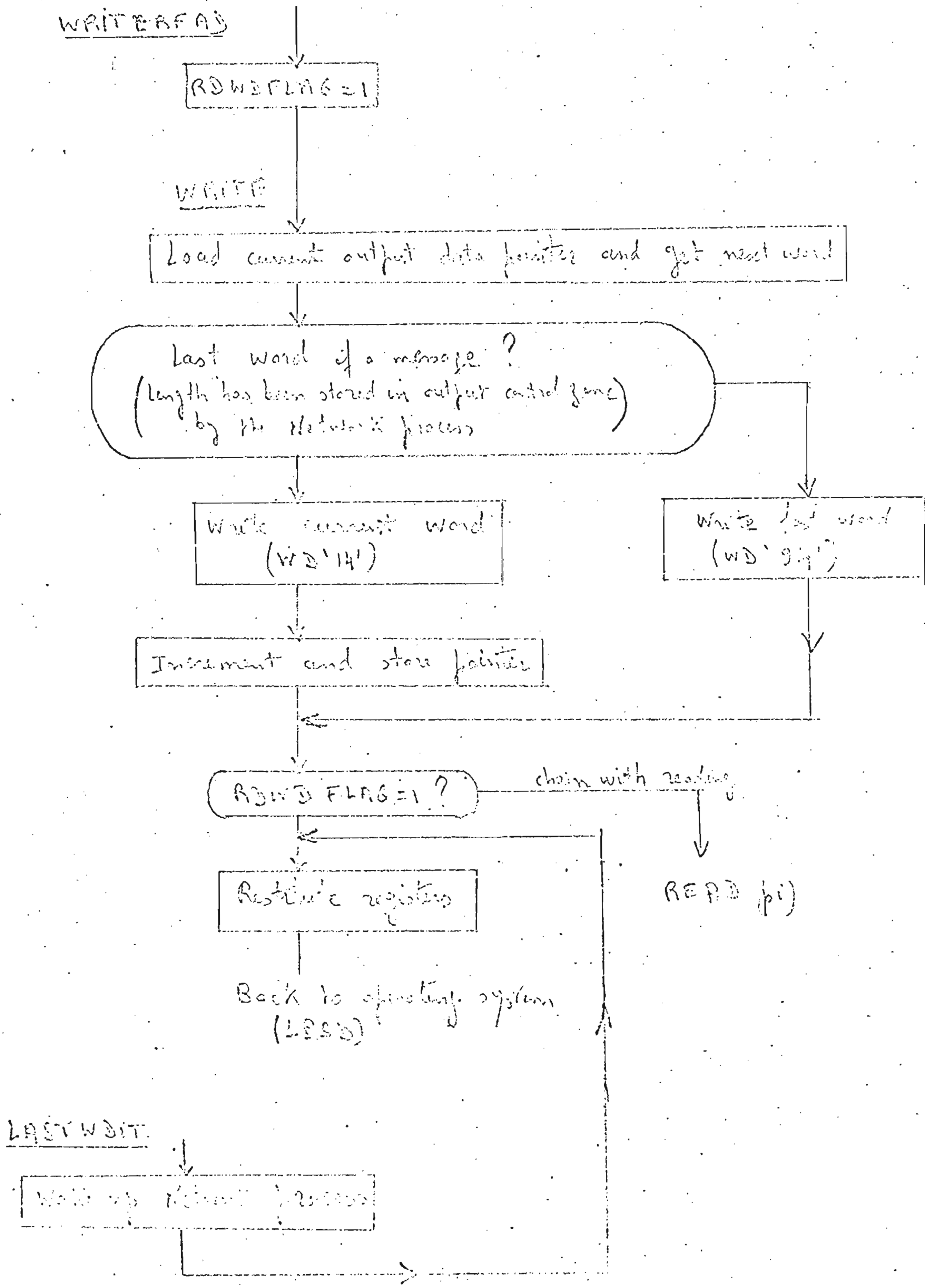

CONTMSG
Parse the first text character of the message
TACK
(Positive ACK)
TNAK (x)
TENQ (x)
TEOT (x)
(x) These options are not flowcharted! They are similar to TACK
TACK
2nd text charact = ENQ ?
yes
Acknowledgment for link establishment
. Select remote HOST # (i) from header
. Select incoming link # (β) from message text
. Select outgoing link # (α) from message text
Retrieve connect # (k)
. Access INPUT-LINK table upon hashing:
"(α) + (i) + outgoing flag"
Update CONNECT table entry (k)
. Turn on "connection established" bit
Update INPUT-LINK table
. Create an entry upon hashing
"(β) + (i) + incoming flag"
. Store in this entry: HOST # (i), incoming link # (β), connection # (k).


RFNM received ?
yes
no
WAKEUP
2nd test charact. = EOT ?
yes
error
Delete entry on CONNECT table
Remove connection shared page
from Network virtual space

Handler. (pii)



Save registers
Determine interrupt cause
(Read interrupt flags (RD))
(*) the same interrupt is used for both read and write operations
READ
WRITE (pii)
WAITREAD (pii)
(Read and write occur at the same time)
LASTWAIT (pii)
(Interrupt following the transmission of the last message word)
READ
Load current input data pointer
Read 1 word (RD)
Last word of a message?
(Test (c))
Store the word in input buffer
Increment and store pointer
Store the word in input buffer
Calculate message length and store it into input control zone
Wake up the Network process.
Restitute registers
Back to operating mode
(LSS's)


WRITEREAD
RDWRFLAG=1
WRITE
Load current output data pointer and get next word
Last word of a message ?
(Length has been stored in output control zone by the Network process)
Write current word (WD'14')
Write last word (WD'94')
Increment and store pointer
RDWRFLAG=1 ?
chain with reading
READ (pi)
Restore registers
Back to operating system (LPSD)
LASTWDIT
Wake up Network process